Jason Edman

CS3310-001

Project 1

Part 1 - Results

Merge Sort



Quick Sort



Merge Sort vs Quick Sort

Part 2 - Questions

Question 1:

Which is generally faster (QuickSort or MergeSort)?  Why (remember why means compare both sides of the proverbial coin)?

In these tests quicksort is generally faster than mergesort. As the arrays grew the count for basic operations of quicksort did increase slightly over mergesort but they were negligible in terms of speed. Quicksort sorts in place needing no extra space to perform sorting while mergesort requires a temporary array to merge the sorted arrays. They both have an average case time complexity of O(nlogn). I included an extra graph above which showed a flaw in my code. Basically, the arrays were already sorted when it got to quicksort which was causing quicksort to take far longer. This was getting times closer to a complexity O(n2) which is the worst case for quicksort. Even though that does show a flaw in quicksort this only applies to rare instances where the array is already sorted, in which case you wouldn’t really need a sorting algorithm or could further randomize the array before sorting. On the other side of things if stability is more or a concern or the array is very large, it might be better to use mergesort, especially if memory is not a concern.

Question 2:

Which generally has fewer basic operations? Why?

Merge sort has fewer basic operations making it slightly more efficient than quicksort. In this dataset the difference is small but as the data set increases in size the gap will widen making mergesort more efficient on larger arrays. In the merge sort function the comparison being made is based on a couple of conditions while the quicksort is only iterating through a set amount (in the partition function). This means that quicksort will always hit a minimum basic count while merge sort can break out if conditions are met. The areas that are affecting this are shown in the code below.

Text

Description automatically generated

A screen shot of a computer

Description automatically generated with low confidence

Question 3:

Were your estimated times consistent (meaning is the actual time close to and does it follow roughly the same time deltas as the actual time did)? Why or why not?

In the case of mergesort the estimated time was very consistent. Through about 5 or 6 different runs the results were always in line. I ran it a couple of times thinking that my arrays were not being randomly generated. This ended up making sense since the best and worst case (and therefore average) time complexities for merge sort are both O(nlog(n)) which is what my estimated time was based off.

In the case of quicksort, I had some mixed results. On a couple of runs the actual times were closer to O(n2). As stated above, this was later figured out to be a code problem, but it still illustrated the worst case for quicksort hence why I still included it with the report. Once the error was resolved the times were much more consistent with the estimated delta for nlog(n). Even though quicksort’s worst case is n2 it still has an average of nlog(n) and overall was even closer to the estimated delta set up for nlog(n).

Proof using best case for merge sort.

1. b(n) = 2\*b(n/2) + n/2; for n > 1 n a power of 2
2. Yes, the recurrence relation holds only for values of n that are powers of 2, i.e., n = 2^k, where k is a non-negative integer
3. Initial constraint is n = 1 or b(1) = 0, basically the input must be an array of 1 and it will return with 0 operations.
4. b(2) = 2\*b(1) + 1 = 0 + 1 = 1; base case is true
5. Assume that the recurrence relation holds for some n=k, i.e. b(k) = 2\*b(k/2) + k/2
6. b(k + 1)=2b((k+1)/2) + (k+1)/2

= 2b(k/2) + (k+1)/2 (since k+1 is odd, we can write (k+1)/2 as k/2 + 1/2)

= 2\*(2b((k/2)/2) + k/4) + (k+1)/2 (substituting b(k/2) with the assumption)

= 4b(k/4) + k/2 + 1

= 4\*b(k/4) + 2\*(k/2)/2 + 1

= 4\*b(k/4) + k/2 + 1

= 2\*(2\*b(k/4) + k/4) + 1

= 2\*b(k/2) + 1

= b(k) + 1

subbing the assumption in

b(k+1) = b(k) + 1

= 2b(k/2) + k/2 + 1

= 2b((k+1)/2) + (k+1)/2

Therefore the relation holds for n = k + 1

By proof of induction the recurrence relation holds for all powers of 2.